Paper ML03

IDD'AI: Automating the Tedious So You Can Focus on the Genius

Lionel Debecq, IDDI, Ottignies, Belgium
Samantha Cambier, IDDI, Ottignies, Belgium
Salah Amar, Euranova, Louvain-la-Neuve, Belgium

ABSTRACT

In the fast-paced world of clinical research, who has time to manually write SAS® programs from mock tables? Enter our AI-powered automation tool, *IDD'AI*, designed to make your life easier and your work faster. This innovative tool aims to use *Natural Language Processing* (NLP) to read mock tables like a pro, picking out the stats you need and variables from the *ADaM metadata*. And here's the kicker: our *LLM* will *rank* potential *matches by confidence level*, so you spend less time hunting for the right code and more time doing what really matters. By automating the boring bits, we aim to help you get the job done *quicker* and with *fewer mistakes*. This *proof of concept* (PoC) by Euranova and IDDI will evaluate efficiency gains, AI accuracy, and regulatory compliance (GDPR, European AI Act) while keeping *human oversight* for reliability. Say goodbye to tedious programming and hello to a smoother, more efficient workflow!

INTRODUCTION

Biostatistical teams often spend excessive time manually translating mock tables into SAS® programs, which can slow down study timelines and sometimes introduce errors.

To address this challenge, we wanted a tool that harnesses AI capabilities and methods to parse mock tables, identify the appropriate datasets and variables from the study's ADaM metadata, and determine the necessary statistical analyses, all from the mock tables. Also, by leveraging our standard macro library, the tool would then use the information gathered to select and populate the corresponding macros, automatically generating SAS® programs tailored to the required statistics.

As we are not experts in AI, we have hired *Euranova*, a company specializing in AI automation projects, to support us on this journey.

GENERAL CONSIDERATIONS

Following the same approach as per ADaM model v2.1 on Figure 1, the usual and familiar approach when it comes to programming outputs is, from top to bottom, the derivation of ADaM datasets, then the production of the results outputs with support of the study documents (protocol, mock tables and SAP, and specifications, depending on the workflow).

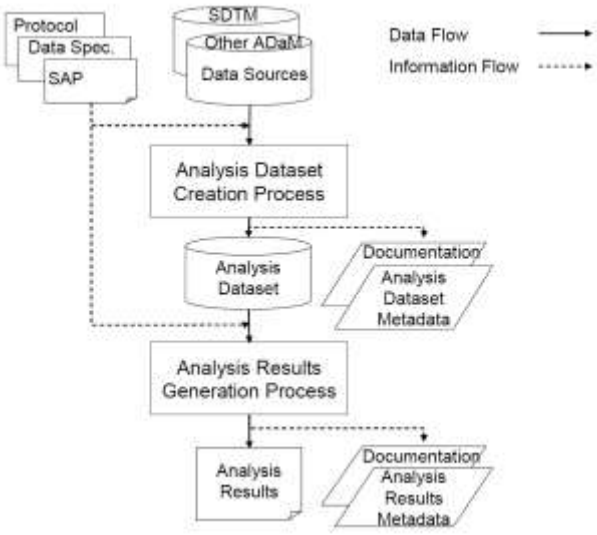

Figure 1 Analysis Data Flow Diagram Showing One Scenario for the Flow of Data and Information (ADaM v2.1 page 8)

While the Analysis Datasets are created with the Statistical Analysis Plan (SAP) and mock tables document, the "Analysis Results Generation Process" relies on the programmer/biostatistician manually matching information from the mock tables to the derived ADaM dataset(s) or study specifications to program the results. Which is a challenge to understand what need to be used from the ADaM datasets, and in terms of time occupation.

To address this challenge, the idea is to leverage metadata generated during ADaM dataset creation and the mock shells, to "automatically" identify the variables that are needed for the programming of the analysis outputs. This method would allow us to identify both variables and the statistics needed and therefore select the appropriate internal macro with pre-filled parameters, streamlining the generation of outputs across all mock tables.

This would also be especially useful for the biostatisticians or junior statistical programmers among us that are not at ease with SAS® or ADaM in general.

Here are the prerequisites:

MOCKS

One of the challenging aspects of this project is parsing the mock tables. Since there are multiple ways to build mock tables, we chose to settle for ".docx" format, in landscape, A4 page size, and, exception for titles and footnotes which are in the page body, the body of the "table" is designed as a table layout, meaning that each column is delimited as a true column in MS Word, and each table header is delimited as a true row. The schema hereafter is representing a typical output in our process where a block of titles is followed by a table block with the statistics and a block of footnotes. All outputs are produced in RTF file format with ODS RTF in SAS®.

The following figure represents the structure of our mocks and, therefore, outputs (Figure 2):

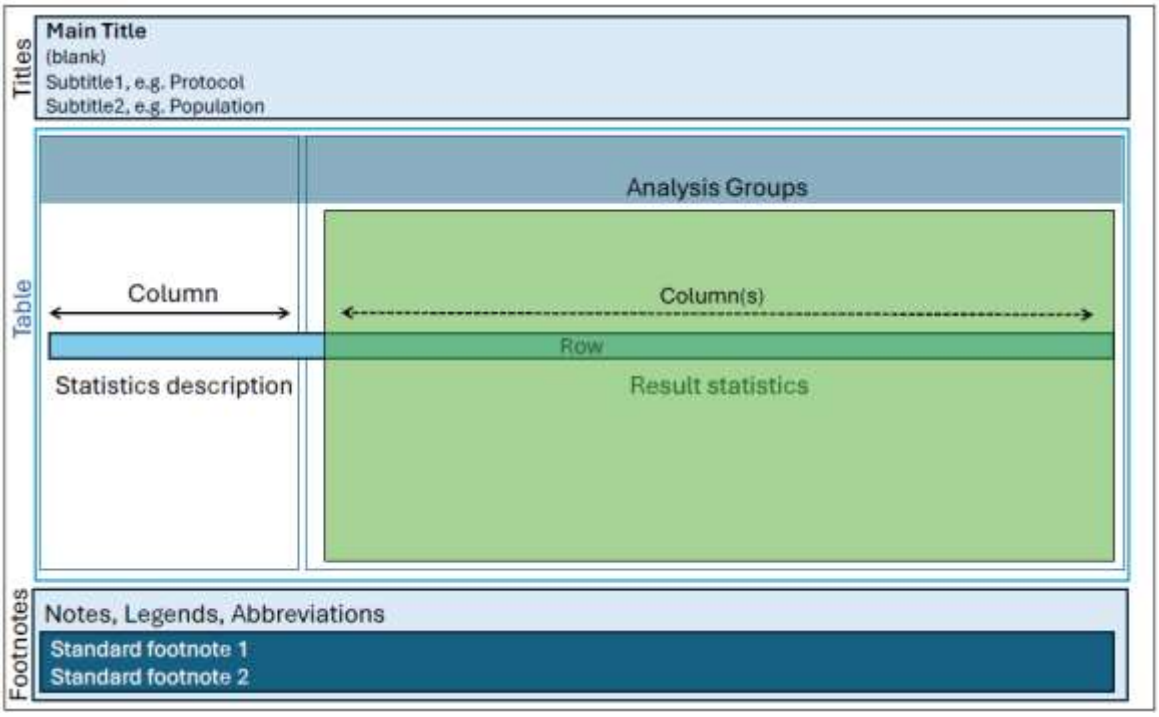

Figure 2 Standard output structure

The basic rules for our mock tables are the following:

1. First line is main title, followed by a blank line, a first subtitle as the study identifier or protocol name as e.g. "Protocol: xxx"; next comes the population analyzed as second subtitle e.g. as "Population: xxx analysis set".
2. The table itself as a table "element" with columns for statistics description and one column for each analysis group. The results will be displayed in the "body" of the table where the result statistics will be printed.
3. A series of footnotes with a maximum of 7 footnotes (notes, legends, abbreviations), 1 blank line of footnote between the footnotes specific to the output and the 2 standard footnotes. Maximum is 10 paragraphs of footnotes.
4. The statistics description MUST have consistent indentation. The indentation could be either a series of spaces or the tabulation character, but they need to be consistent for the "level" considered and throughout the mock.

e.g.,

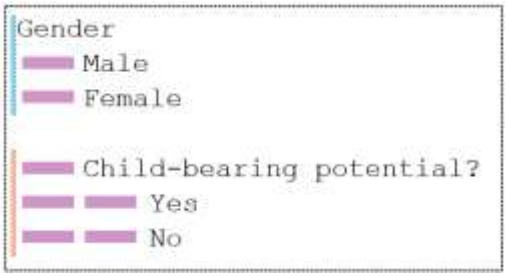

Figure 3 Indentation consistency

In this visual example (Figure 3), the blue and orange are referring to two different variables as their values are under their "title" with the adequate number of indentations (in purple). Although, the second one is not dependent on the previous indent (gender), except for percentages calculation if required.

For this proof of concept, we focused only on Baseline and Safety mock tables.

ADAM METADATA

All metadata are generated during programming of the Derived ADaM Datasets through our ADaM Macro that collects all required information and stores them into SAS® datasets. Originally, those metadata are used by an internal tool, "ID-Spectackle", to complete, maintain and generate Study Specifications before generating the Define-xml with Pinnacle 21. The metadata are converted into JSON files with (1) Dataset metadata: dataset name, dataset label; (2) Variable metadata: variable name, variable label, variable type, codelist name for variable if applicable; (3) Codelist metadata: codelist name, codelist code and decode. The JSON file is generated from SAS® Datasets, by SAS® with PROC JSON, during ADaM Dataset finalization phase of the Derived Datasets programming. The result of the Metadata in JSON is represented by this example,

MACROS LIBRARY

One of the key features of the project being the automated creation of analysis output programs, a library of standardized macros is required to be able to manage as easily as possible the outputs. For this exercise, we will focus on a descriptive statistics macro (mean, median, etc.) and one categorical statistics macro (frequencies), as well as a second categorical specific to occurrence data (as presentation in output is a lot different, we chose to have a second one specific for occurrence data e.g. AE per SOC/PT). The function of these standard macros are (1) to calculate the statistics; (2) format the results for proc report.

METHODS

This project can mainly be divided into 4 key points:

1. Information Extraction from mock tables: Parsing and identification of key elements (titles, footnotes, rows, columns, type of statistics)
2. Building a Mapping Dictionary (from mock tables and study metadata): Mapping of mock contents to available metadata.
3. Variable Classification: Detection of statistics and selection of appropriate standard macro from library
4. Automatic SAS® Program Generation: Generation of programs from template with prefilled information and basic selection.

INFORMATION EXTRACTION FROM MOCK TABLES

The Information Extraction from mock tables relies on a rule-based module for parsing the mock tables, with the support of an AI for linguistic and syntax correction.

This module's primary goal is to programmatically parse Word ".docx" files to locate and extract structured data from tables. It's implemented as a deterministic, rule-based Python application that relies on the *python-docx* library.

The extraction workflow functions as a linear pipeline that walks through a Word document from start to finish. It iterates over core elements -paragraphs and tables- and leverages a state machine to distinguish between table metadata (variables, labels, …), table contents (statistics), and surrounding text annotations (titles and footnotes).

The general workflow for parsing mock tables document is as follows:

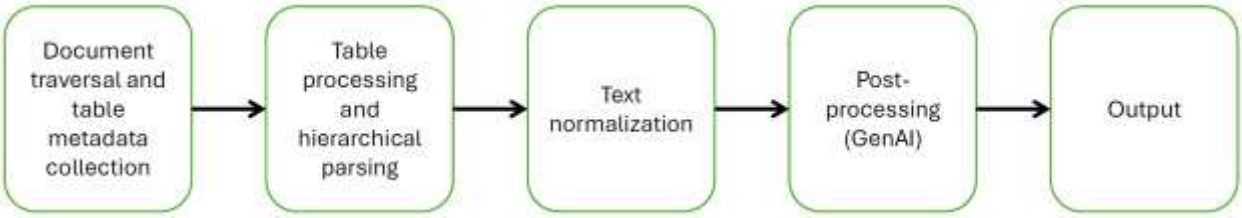

Figure 4 General workflow for parsing mock tables document

1. DOCUMENT TRAVERSAL AND TABLE METADATA COLLECTION

All body elements from the docx are processed in order by the Python script, and managed by a simple two-state machine:

- State “LOOKING_FOR_TITLE”: In this initial state, the function scans each paragraph. If a paragraph's text matches a predefined regular expression, it signifies the start of a new table. The script then extracts the id and title of the table, stores any preceding text in a buffer, and transitions to the GATHERING_METADATA state. Any text that does not match is appended to the “notes” buffer.
  State “GATHERING_METADATA”: After a title is found, the script collects subsequent lines as potential metadata, such as the study protocol information, or population. When a table element is encountered, the script proceeds to parse its contents.

2. TABLE PROCESSING AND HIERARCHICAL PARSING

The Table Identification is done through regular expressions to detect titles, using them as markers for new table entries. Upon finding a “Table” element, the function “GATHERING_METADATA” passes the table object to the “Hierarchical Variable Parsing” function, which is responsible for interpreting the structure within a table body. This function iterates for all rows within a table element.

Titles are matched as the first line on the mock, while between the title and the table element itself is considered as subtitles (or notes). All this information is saved as table metadata.

“Hierarchical” means it relies on proper and consistent indentation within a block of data in the table element. Once a table is found, each row is inspected, with indentation in the first cell used to build parent-child relationships among variables. (e.g., Gender-Male/Female)

A non-indented row is classified as a new primary variable (parent variable).
An indented row is classified as a sub-category or content of the most recently identified primary variable and is appended to the content of that primary variable.

For each block of data, there is a basic heuristic in place to predict the statistic. When a “%” sign is found on the row, it is categorical; while Mean or Median implies continuous. When neither is present, the type selected by default is categorical.

Of note, although the parsing process went relatively well overall, our test data files contained inconsistencies between the mock tables -such as differences in formatting, indentation, and “non-standard” outputs requested by the sponsor. Based on our experience with this project, we strongly recommend keeping mock tables as standardized as possible to maximize parsing accuracy and improve the matching to ADaM variables.

3. CLEANING AND TEXT NORMALIZATION

This step has the only purpose to clean a predefined list of suffixes from the end of a string (e.g., *(x, n=, %), ;, ,*), and cleans the string processed from the mocks for easier processing and subsequent steps.

4. GENERATIVE AI AND POST-PROCESSING

Although the parser gave a good basis, it was necessary to use a generative AI, with carefully tailored prompts, to refine and correct some parts of the resulting JSON output from the Parser. This pipeline leverages a large language model (LLM), Google Gemini (through direct API call) to perform tasks that require contextual understanding beyond simple structural rules. Each step involves sending the extracted data to the model with a specific set of instructions and receiving a modified, structured JSON object in return.

For straightforward tasks, we use *zero-shot prompting,* role play prompting (for example, “You are a Clinical Data Analyst AI”) and we provide detailed instructions. For more complex tasks like fixing fragmented text, we switch to *few-shot prompting* by including clear input–output examples. The model temperature is set to zero to ensure deterministic results and minimize hallucinations. Finally, we lock the response format to a predefined JSON schema, ensuring deterministic, machine-readable output.

One other important task of the generative AI post-processing, is to identify “repeats”, should they be table or content in a table.

- Repeated Table: The model scans the “notes” (from the table metadata) for sentences like “Similar to” to establish a relationship between the current table and another one in the document. Also, a repeat table does not have any “Table” element after its title.
- Repeated Logic (or content): The model scans for “Repeat for each” to capture when an analysis should be repeated across subgroups (e.g. by parameter, by treatment group, etc.). The model is specifically instructed to ignore simple time-based repeat such as “Repeat for each month”.

5. OUTPUT

The final output of the extraction module is a clean, machine-readable representation of the mock tables. All elements as saved as a JSON array. Each element in the array is an object representing a single table from the source document.

MATCHING PARSED MOCKS TO ADAM METADATA

This aims to map the extracted data from the Mocks to the Study Metadata, that provides all the necessary information for the matching, e.g. variable names, variables labels, etc.

Without a large, labeled dataset for training or fine-tuning a model, we relied on unsupervised and zero-shot strategies to tackle this low-resource matching problem. We experimented with three approaches: direct prompting of a Large Language Model (Gemini), fuzzy string matching, and semantic search using a sentence-transformer.

The rule set was developed based on five annotated documents containing mock tables entries, hereafter referred to as *variables*. A total of 66 variables related to demographic and safety data were extracted for rule development. Validation was performed using 45 entries from a single study-specific document. As the procedure is deterministic, a single test iteration was sufficient, with repeated testing expected to yield identical results.

- APPROACH 1: LARGE LANGUAGE MODEL (LLM)

A detailed prompt has been crafted and presented to the LLM with each extracted variable alongside a complete ADaM metadata JSON. The model -acting as a “data mapping expert”- was asked to pick the single best matching ADaM variable and its description for every primary variable.

A manual, qualitative review against the unlabeled dataset uncovered frequent hallucinations: the model invented matches that didn’t exist in the metadata. This unreliability made the zero-shot prompt approach unsuitable for a production-quality data pipeline.

- APPROACH 2: FUZZY STRING MATCHING

The second method employs algorithms that assess lexical similarity, matching strings based on their character-level structure. It relies on *rapidfuzz* library, to calculate a similarity score between two strings based on the number of insertions, deletions, or substitutions needed to make them identical.

The implementation is as follows:

- Similarity Matrix Construction: Character-level similarity scores between extracted variables and ADaM metadata descriptions were computed using algorithms such as *rapidfuzz.QRatio*. To improve discriminative power, multiple similarity matrices were generated by comparing source variables to various target features (e.g., acronyms, full descriptions).
- Global Assignment Optimization: Instead of a greedy, pairwise matching strategy, the Hungarian algorithm was applied to identify the optimal one-to-one mapping between variables. This combinatorial optimization approach maximizes the aggregate similarity score across all assignments, thereby minimizing suboptimal or ambiguous matches.

Being purely lexical, the model lacked semantic understanding. Its primary weaknesses were:

- Inability to Handle Synonyms: It could not recognize that terms like "subjects" and "patients" or "discontinuation" and "withdrawal" were semantically equivalent.

- Confusion with Similar Phrasing: It was easily confused by variables that looked alike but had critically different meanings, such as failing to distinguish between a "*TEAE*" (Treatment-Emergent Adverse Event) and a "*Serious TEAE*".

Overall, the fuzzy matching approach proved too brittle and was not robust enough to handle the nuances of clinical terminology.

- APPROACH 3: SENTENCE TRANSFORMER EMBEDDINGS

This approach leverages sentence embeddings, where in a deep learning model encodes textual data into high-dimensional vectors. The principal advantage of this representation is that semantically similar texts are mapped to proximate points within the embedding space.

This method uses a pre-trained sentence transformer model (*all-miniLM-L6-v2*) to perform semantic matching:

- Text Encoding: The transformer model is used to generate a vector embedding for every extracted mock description and every target ADaM metadata description
- Semantic Similarity: Cosine similarity is calculated between every embedding, that produces a similarity matrix where each score represents semantic closeness rather than lexical similarity.
- Top-K Candidate Selection: For each extracted variable, the system identifies the top ADaM variables with the highest cosine similarity scores, yielding a ranked list of the most probable matches.

- Conditional Refinement Heuristic: To address generic or context-dependent variables (e.g., "Baseline", "Change from Baseline", "Value"), a rule-based refinement is applied. If the initial match corresponds to a generic ADaM variable (such as BASE or CHG), the system discards this result and recalculates similarity using the more descriptive table title as the source text. This two-pass approach enables more accurate mapping of generic terms to their specific contextual counterparts (e.g., mapping "Baseline" from "Summary of Systolic Blood Pressure" to the SYSBP baseline variable).

To assess the correctness of the approach, annotated data and non-annotated data were used.

- Qualitative Assessment (Unlabeled Data): On the test dataset, the proposed method outperformed previous approaches. The identified matches corresponded more closely to human judgment, and even incorrect predictions tended to involve semantically related concepts, suggesting a more nuanced understanding of the underlying text.
- Quantitative Assessment (Labeled Data): Using a labeled dataset of 66 mock tables, the model's performance was quantitatively evaluated.
    - *Top 1 Accuracy*: 60% (the correct label was the model's highest-ranked prediction in 60% of cases)
    - *Top 3 Accuracy*: 89% (the correct label appeared among the top three predictions in 89% of cases)

These results indicate strong potential for the method, both in terms of alignment with human intuition and formal accuracy metrics.
This method comes with some well-known limitations and trade-offs too. One of them is that most sentence transformers are based on BERT or similar architectures with a fixed input length (typically 512 tokens). This approach, unlike the first one, has limited context window. Which means that long documents or sentences may not have the desired result.

Of course, other limitations are present for this method (domain specific models, fixed-length embeddings limited to a certain length, etc), although this is the preferred approach, considering the good results compared to the two other approaches for the matching label.

VARIABLE CLASSIFICATION

This step is done during information extraction from mock tables, at the table processing step.
The classification is done through simple conditional tests:

- If the "result" columns contain "%" sign or can't be determined, the statistic is categorical
- If the "result" columns contain either Mean or Median, the statistic is descriptive

"AUTOMATIC" SAS® PROGRAM GENERATION

The other purpose of this project, beyond automatically associate mock entries to ADaM variables, is to determine which statistics are required in a particular table and, doing so, pre-complete our programs accordingly. Statistics detection have been done during the parsing step. With the dataset name, variable name, mock entry name and the type of statistics required, we were able to pre-complete some part of the reporting program.

The parser was able to detect, to a certain extent, which statistic is needed for the mock content and associate it with the ADaM variable.

Our process includes a series of standardized macros that perform different statistics and format them for proc report and output in RTF format: *freq-tab* (for categorical statistics), *univ-tab* (for descriptive statistics), and *events-freq* (for occurrence data, e.g. Adverse events, concomitant medications, etc.).

The tool programmatically completes the relevant macro calls and integrates them into a standardized SAS® program from the identification of the type of statistics and variables matched at the previous step. The resulting output closely approximates the final production-ready program, requiring only minor adjustments to the data selection logic by the Programmer or Biostatistician, and adaptation of the macro parameters if not as required completely.

This is achieved by *Jinja*, a template engine package in Python under BSD License (Berkeley Software Distribution), that allowed us to use our standard programming templates and complete the necessary parts from the previous steps by adding tags where the tool must add the data selection and macro call.

The end-product is one SAS® program file per table found in mock tables document, with basic data selection and calls to our standard macro in a sequential order (i.e. order of the statistics on the mock table). Although, noted that no programming optimization is done and some programs might be repeated with different population or selection on the same data (e.g. adverse events tables). Some data transformation might be required, such as transpose, etc. At stage of writing, only integration of categorical and descriptive statistics macro is done, macro for occurrences is still to be integrated. Data selection is very basic and only allows simple where clause and keep variables analyzed, dependency on other variables such as flags is not (yet) included.

DISCUSSION

WORKFLOW

The tool has been developed in Python (v3.11.5) and has been implemented as a web-based application, enabling seamless deployment within a Docker container on the Microsoft Azure cloud platform.

Our current workflow entails manual inspection of mock outputs and derived datasets to identify relevant variables and parameters. This is followed by the configuration of programs, selection criteria, and macros necessary to generate the final RTF outputs.

By integrating the tool immediately after the derivation of ADaM datasets, we aim to streamline this process significantly. The tool automates many of the labor-intensive steps, thereby reducing manual effort and allowing the team to reallocate focus toward higher-level tasks such as refining study specifications and ensuring methodological rigor.

BENEFITS

1. Increased Efficiency

Automation of Repetitive Tasks: The tool automates the generation of programs and macros needed for RTF output, reducing the time spent on manual setup.
Faster Turnaround: By bypassing manual mock reviews and dataset exploration, analysts can move more quickly from ADaM dataset derivation to output generation.

2. Improved Consistency and Reproducibility

Standardized Output Generation: Automated processes reduce variability in programming practices, ensuring consistent formatting and logic across studies.
Reduced Human Error: Minimizing manual intervention lowers the risk of coding mistakes or inconsistencies in derived outputs.

3. Enhanced Focus on Scientific and Strategic Tasks

More Time for Interpretation: Analysts can allocate more time to interpreting results, refining study specifications, and addressing scientific questions.
Support for Methodological Development: Freed-up resources can be redirected toward improving statistical methods or exploring innovative analysis strategies.

4. Scalability and Flexibility

Cloud Deployment: Hosting the tool on Microsoft Azure via Docker containers allows for scalable use across teams and studies, with minimal infrastructure overhead.
User-Friendly Interface: The intuitive UI facilitates adoption by users with varying levels of technical expertise, promoting broader use within multidisciplinary teams.

5. Seamless Integration with Existing Standards

Alignment with CDISC Standards: The tool fits naturally into workflows involving ADaM datasets, supporting regulatory compliance and submission readiness.

6. Standardized Macro Usage

Promotes the use of validated and reusable macros across studies, improving code maintainability and reducing duplication of effort.

CONCLUSION

The initial objective of this project was to develop a comprehensive library of ADaM metadata that would capture the relationships between variables and mock table content, thereby enabling predictive mapping for future studies. However, due to the limited availability of proper ADaM metadata (a lot of available studies were IDMCs or not fully ADaM) and the high variability in both implementation of the standard and mock table layout, this approach was deemed impractical and subsequently abandoned.

The rule-based approach proved to be a decent and reliable approach, while the need for AI involvement was limited to small correction inherent to table formatting. The matching and scoring system were robust enough to allow anyone to select the appropriate variables from the metadata and therefore obtain a usable program with limited edition required. While AI won't replace the human (yet), the automation of some tasks such as spotting what variable for what statistic in a mock table help (1) the development of output programs in a timely manner; (2) to process the data with less error due to incorrect data selection from the user; (3) the employees without or limited knowledge with the programming language of choice.

This proof of concept focused exclusively on SAS® programs generation. By enforcing strict rules for macro completion, we were able to generate functional programs that simplified certain programming tasks. However, manual intervention remains necessary, particularly for data selection scenarios that cannot be reliably inferred from mock table content alone -such as the application of analysis flags or parameter-specific criteria. Additionally, the generated programs are relatively basic and may lack optimization. In some cases, they are not directly usable due to required data transformations (e.g., transpositions, etc) or layout constraints that exceed the capabilities of the available standard macros.

We unfortunately met roadblock with the mocks due to lack of standardization. The need for standardized mock table formats is critical. In response to this challenge, we initiated the development of a library of mock tables that adhere to internal formatting standards aligned with the capabilities of our macro library. This alignment ensures consistency in layout and facilitates seamless integration with automated programming workflows.

If this Project finds itself useful in the long term, we might investigate figures and listings, as well as efficacy results.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Lionel Debecq
Company: IDDI S.A.
Address: Avenue Provinciale, 30 1341 Ottignies-Louvain-la-Neuve
Work Phone: +32 10 614 444
Email: lionel.debecq@iddi.com
Website: https://www.iddi.com

Author Name: Samantha Cambier
Company: IDDI S.A.
Address: Avenue Provinciale, 30 1341 Ottignies-Louvain-la-Neuve
Work Phone: +32 10 614 444
Email: samantha.cambier@iddi.com
Website: https://www.iddi.com

Author Name: Salah Amar
Company: Euranova
Address: Rue Emile Francqui, 4 1435 Mont-Saint-Guibert
Work Phone: +32 10 75 02 00
Email: salah.amar@euranova.eu
Website:http://www.euranova.eu